MONASH University

A/Prof Lindsay Kleeman
Clive Maynard

ECE3073 / TRC3300
Computer Systems

Mutual Exclusion

WARNING

COMMONWEALTH OF AUSTRALIA
Copyright Regulations 1969

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

MONASH University

Inter-process Communication, Synchronization, and Deadlocks

- Why do we need Cooperating Processes?
- Interprocess Communication
- Synchronization
- Deadlocks

Protection Builds Walls

- Need high-performance systems
- Need multiple processes
- Multiprocessing requires isolation
- Our processes need communication
- Sometimes separate machines

Why do we need Multiple Processes?

- Performance
- Scaling
- Purchased components
- 3rd party service
- Components in multiple systems
- Reliability
- Physical location of information
- Enable application

Communication Mechanism Attributes

- Number of processes
- One-way or bi-directional
- Buffering strategy
- Connection oriented or connectionless
- Naming strategy
- Multicast, broadcast, unicast
- Number of connections
- Streaming or message oriented
- Heterogeneous or homogeneous
- Synchronous or asynchronous
- Persistent or Transient

Many Types of Communication

- We don’t have time in this unit to go into detail of the many communications approaches.
- We do need to understand synchronisation between processes!

Synchronization

- Proc A puts record in buffer
- Increments counter X

  X = X + 1
- Proc B gets record from buffer
- Decrements counter X

  X = X - 1

Mutual Exclusion Example

- **Assume context switch possible at *any* time.**
- **Standard example**

```
void task1() {
 while (1) {
      temp = shared_var;
      res = update1(temp);
      shared_var=res;
 }
}
```

```
void task2() {
 while (1) {
      temp = shared_var;
      res = update2(temp);
      shared_var=res;
 }
}
```

Mutual Exclusion Example

- **Assume context switch possible at *any* time.**
- **Simple example to illustrate why shared memory needs protection with a critical section**

Process 1	Process 2
C code	C code
var++;	var-- ;

If code protected with critical section *var* will be unchanged after process 1 and 2 execute.

Mutual Exclusion Example

- **Assume context switch possible at *any* time.**
- **Simple example to illustrate why shared memory needs protection with a critical section**

Process 1		Process 2	
C code	Assembler	C code	Assembler
	r2 < - Memory[var]		r2 < - Memory[var]
var++;	increment r2	var-- ;	decrement r2
	Memory[var] < - r2		Memory[var] < - r2

If code protected with critical section *var* will be unchanged after process 1 and 2 execute.

MONASH University

What goes wrong?

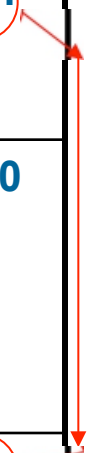
Proc	OS view	Assembler code	var	r2
1	Context switch to process 2, r2 saved in process descriptor table	r2 < - Memory[var] increment r2	10	
2	context switch to process 1, r2 restored from proc. desc table	r2 < - Memory[var] decrement r2 Memory[var] < - r2		
1		Memory[var] < - r2		

Whoops! var has been corrupted (should have been 10)

MONASH University

What goes wrong?

Proc	OS view	Assembler code	var	r2
1			10	
		r2 < - Memory[var]		10
		increment r2		11
	Context switch to process 2, r2 saved in process descriptor table			
2		r2 < - Memory[var]	10	10
		decrement r2		9
		Memory[var] < - r2	9	
	context switch to process 1, r2 restored from proc. desc table			
1				11
		Memory[var] < - r2	11	



Whoops! var has been corrupted (should have been 10)

Locks
aka semaphores and mutual

```
main()
{
   entry section        /* check lock free */
   critical section     /* change shared data */
   exit section         /* show lock free */
   remainder section    /* everything else */
   }
```

Critical Section with a Semaphore

```
// declarations needed in uC/OS -II RTOS
OS_EVENT *ProtectSem;
INT8U err_protect;
….
// initialisation of semaphore and semaphore counter starts at 1
ProtectSem = OSSemCreate (1);
….
// critical section in process 1
OSSemPend (ProtectSem, 0 , &err_protect);
    // code in critical section here eg. Var++
OSSemPost (ProtectSem) ;

// critical section in process 2
OSSemPend (ProtectSem, 0 , &err_protect);
    // code in critical section here eg. Var --
OSSemPost (ProtectSem) ;
```

MONASH University

CS Macro - Making it easy to use

```
// MACRO definition CS() follows …
#define CS(x) OSSemPend (ProtectSem, 0 , &err_protect); x; OSSemPost
    (ProtectSem)

// declarations needed in uC/OS -II RTOS
OS_EVENT *ProtectSem;
INT8U err_protect;
….
// initialisation of semaphore and semaphore counter starts at 1
ProtectSem = OSSemCreate (1);
….
// critical section in process 1
CS(var++);

// critical section in process 2
CS(var --);
```

MONASH University

- In Practice:

- Mutexes must be created before they are used. (a no brainer but it does happen!!)

- The system should not suspend the task owning the mutex when any other task needing it has a higher priority
  …..See Priority inversion